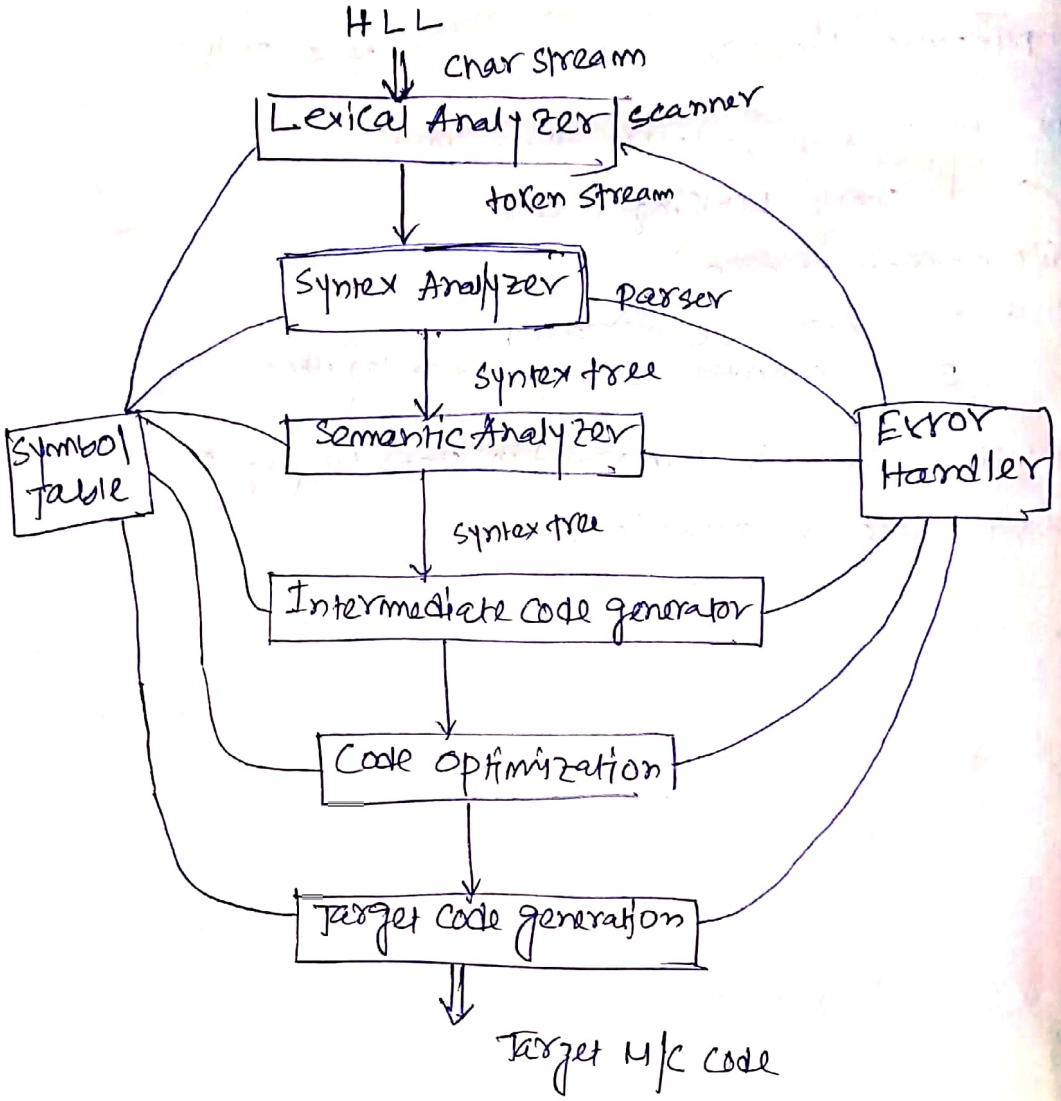
COMPILER DESIGN

Question Asks Before going to study compiler Design

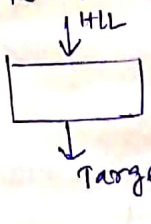
(1) What are the different levels of programming Languages



- HLL program converted to target M/C Code.
- Different Different phases are present inside compiler
- Symbol table used here to store & retrieve information
- In whole procen from HLL to target M/C code if any phase ~~requir~~ wants to store information in the time of procning and also wants to retrieve, ~~them~~ for that purpose symbol table is required.

- Error Handler is useful for Handle error that occure in any phase.

- single pan compiler:



- Two major phases

concentrate on language (HLL) → Analysis phase → (M/c independent / Lang. dependent)

concentrate on Machine Level Language → synthesis phase → (M/c dependent / Lang. independent)

phase 1: Lexical Analysis

Reads the stream of characters making up the source program & group the character into meaningful sequences called lexeme.

Lexical Analyzer represents these lexems in the form of tokens

< token-name, attribute-value >

How to programs converted into token

Example: new_value = old_val + 12

tokens: new val identifier

= Assignment operator

old_val identifier

+ Add operator

12 Number

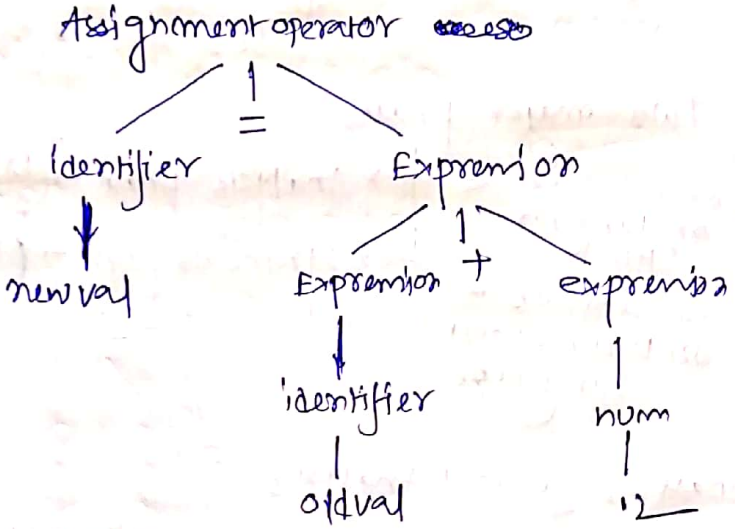
** Remove spaces.

Phase 2: Syntax analysis

also called parsing.

- It takes token produced by lexical as input and generate a parse tree or syntax tree.
- Example:

newval = oldval + 12



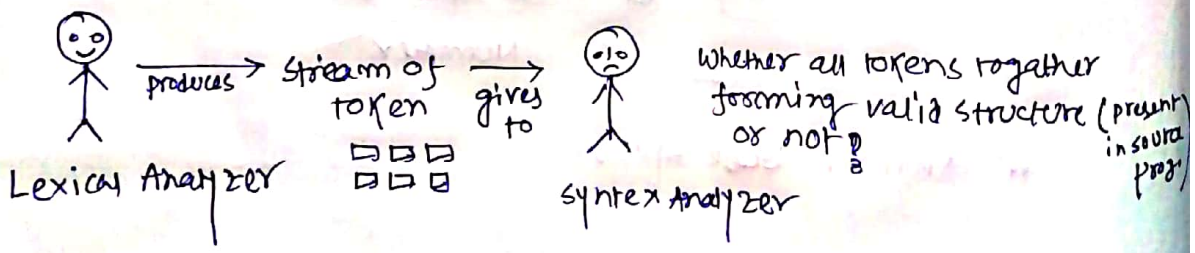
Syntax of a language refers to the structure of valid programs / statements of that language

- specified using certain rules → known as productions
- collection of such productions (rules) is known as grammar.

parsing or syntax Analysis

↳ is a process of determining if a string of tokens can be generated by a grammar.

Task 1: parsers / syntax Analyzer gets string of tokens from Lexical Analyzer and verified if that string of tokens is a valid sequence i.e whether its structure is syntactically correct.



Other Tasks of A parser

(Task 2) - Report syntactic errors

(Task 3) - Recovery from such errors so as to continue the execution procm.

output of Parser

- A representation of parse tree
- representation is generated using the stream of tokens provided by the Lexical Analyzer

#. Type Checking also perform in parsing

GRAMMARS

"Grammar basically a set of rules called productions that define the valid structure of a particular Language"

{ What are the components of Grammar &
~~How to derive a particular string (valid) of a language~~
How to derive a valid string from a grammar of a particular language.

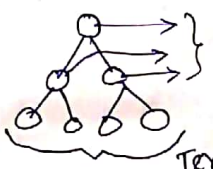
A grammar consists of 4 components

1. Set of Tokens / Terminals (End point)
2. Set of Non Terminals (components that are replaced by terminal)
3. Set of productions

<Left Hand Non terminal> → <Right Hand Terminal/Non terminal>

NT, followed by arrow, followed by seq. of T and/or NT (RHS)

if we represent terminals & Non terminals by tree,



Non terminals

Terminals

S → PQ
P → P
Q → q

} productions

4. Start symbol: Begining of any derivation.

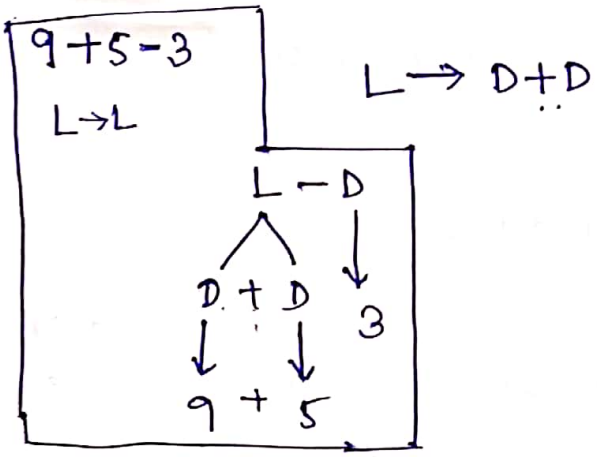
A grammar is specified by listing its productions, with the production for the start symbol appearing first.

Example: All the expression with single digit having either '+' or '-' between them

$L \rightarrow D+D \mid D-D \mid D$ ⟹ This is currently incomplete

$D \rightarrow 0|1|2|3|4|5|6|7|8|9.$

9+5



How to determine strings from the grammar.

⇒ Start with the productions having the start symbol on the LHS

⇒ Repetedly replace all the NT (on RHS) by their productions.

- All the strings that can be derived from grammar belong to the Language specified by that grammar.

Language ⟶ Grammar (Rules) ⟶ productions

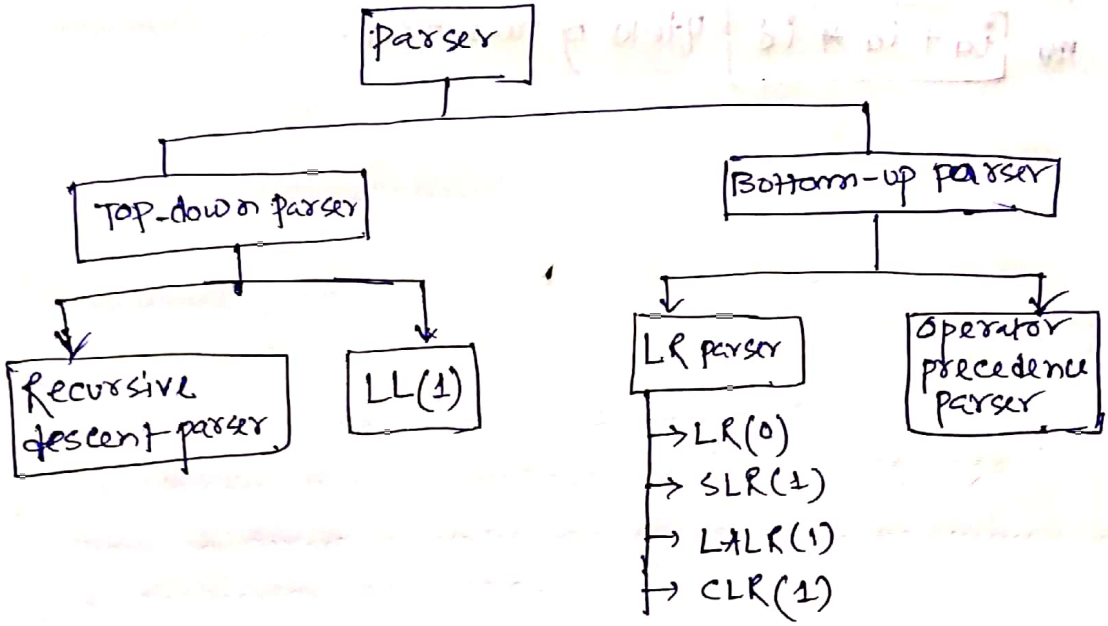
NT T

(one NT will be start symbol (S))

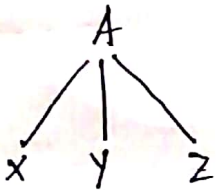
PARSE TREES

Parser is the phase of compiler which takes token string as input and with the help of existing grammar, converts it into the corresponding parse tree. parser is also known as Syntax Analyzer.

* The way the production rules are implemented (derivation) divides parsing into two types : top-down parsing and bottom-up parsing.



* Root Node of parse tree has the start symbol of the Given grammar from where the derivation proceeds.
* Leaves of parse tree represents terminals
* Each interior node represents productions of grammar.
* If $A \rightarrow xyz$ is a production, then the parse tree will have A as interior node whose children are x, y and z from its left to right.



Construct parse tree for $E \rightarrow E+E \mid E*E \mid id$

construct parse tree for

$$S \rightarrow SS* \mid SS+ \mid a$$

Yield of parse Tree:

Leaf nodes of parse tree are concatenated from left to right to form the input string derived from a grammar which is called yield of parse tree.

Ex: **id + id * id** yield of parse tree.